

# Programowanie dla (przyszłych) inżynierów

A. Matuszak

4 października 2010

Wersja 2010

Wersja: 4 października 2010

# Rozdział 1

## Praca z programem

### 1.1 Kalkulator

Na początek warto popatrzeć na OCTAVE jak na kalkulator, przecież tak naprawdę będziemy używali OCTAVE do obliczeń.

Po uruchomieniu programu, można wpisać  $2+2$  nacisnąć enter (zamiast klawisza  $=$ ) i zobaczyć ile wynosi wynik. Ten sposób użycia jest wygodniejszy niż wciskanie klawiszy kalkulatora, gdyż zanim zatwierdzimy wyrażenie do obliczenia (poprzez naciśnięcie Enter), widzimy co wpisaliśmy i trudniej się pomylić<sup>1</sup>.

Kalkulator dysponuje czterema działaniami arytmetycznymi:  $+$ ,  $-$ ,  $/$ ,  $*$ , podobnie mamy te cztery działania dostępne w OCTAVE. Możemy np. napisać:  
 $2*3+5*7$

a OCTAVE wypisze wynik. Znaki odstępu są ignorowane, więc równoważne są zapisy wyrażenia:  $2*3+5*7$  lub  $2*3 + 5*7$  albo  $2 * 3 + 5 * 7$ .

Zapis działań jest bardzo podobny jak na kartce ale są pewne subtelne różnice. Najważniejsza dotyczy dzielenia.

Kolejność wykonywania działań jest taka, jak w matematyce, czyli najpierw mnożenie i dzielenie, potem dodawanie i odejmowanie, ale w przypadku programu, zapis często utrudnia nam ustalenie, w jakiej kolejności działania będą wykonywane.

W przypadku wyrażenia  $2/3+5$  zostanie to obliczone jako  $\frac{2}{3} + 5$  natomiast łatwo się pomylić i chcąc obliczyć  $\frac{2}{3+5}$  wpisać takie wyrażenie. W przypadku zapisu na kartce mamy ładną kreskę ułamkową, która mówi nam o kolejności działań. W przypadku kreski ukośnej takiego ułatwienia nie mamy i dlatego często powinniśmy wymuszać właściwą kolejność działań.

Przy zapisie dzielenia w postaci ukośnej kreski  $/$  pojawiają się niejednoznaczności. W przypadku wyrażenia  $2/3*2$  nie jest jasne, czy miało to być  $\frac{2}{3*2}$ , czy też  $\frac{2}{3} \cdot 2$ .

---

<sup>1</sup> Nie od rzeczy będzie też wspomnieć o możliwościach edycji tego co wpisujemy, jak też o historii, czyli przywracaniu poprzednio wykonanych linii poprzez strzałki w górę i w dół.

W OCTAVE istnieje możliwość (a nawet czasami konieczność) użycia nawiasów okrągłych ( ) do grupowania wyrazów i wymuszania kolejności wykonywania działań. Nawiasy można dowolnie zagłębiać, czyli wewnątrz jednych nawiasów użyć drugih. Jednak w tym celu można użyć tylko nawiasów okrągłych, nie wolno używać (przynajmniej do grupowania) kwadratowych i wąsastych.

Nawiasów może być więcej niż to niezbędne:

$((1/2)+(3/2))/5$

Każdy nawias otwierający musi być zamknięty. W przypadku omyłkowego zapomnienia któregoś nawiasu, np.:

$((1/2)+(3/2)/5$

OCTAVE pokazuje znak zachęty `>` oczekując na zakończenie wyrażenia. Jest to dość stresująca sytuacja dla początkujących ale zorientowawszy się w przyczynie, wystarczy wpisać brakujący nawias<sup>2</sup>.

OCTAVE wymaga jawnego wpisywania znaku mnożenia w postaci `*`, chociaż w matematyce najczęściej jest on pomijany, wystarczy napisać  $ab$  i wiadomo, że chodzi o  $a \cdot b$ . To ograniczenie wynika z faktu, że OCTAVE nie jest w stanie się domyślić, że pisząc  $1\ 2$  mieliśmy na myśli  $1 \cdot 2$ . Ten problem jeszcze bardziej jasnowo uwidacznia się w przypadku zmiennych (o których za chwilę), gdyż w matematyce używamy zwykle jednoznakowych nazw, podczas kiedy w OCTAVE często używamy wieloznakowych. W konsekwencji zapis  $ab+1$  nie jest jednoznaczny, czy chodzi o  $a \cdot b + 1$ , czy też istnieje zmienna o nazwie  $ab$  i chcieliśmy do niej dodać jeden.

Oprócz czterech opisanych działań arytmetycznych mamy jeszcze potęgowanie. Aby obliczyć  $2^5$ , ponieważ nie da się z klawiatury wpisać indeksu górnego, OCTAVE oferuje specjalny operator `^`, który jest odpowiedzialny za potęgowanie. Tak więc  $2^5$  obliczymy:

$2^5$

Tradycyjnie przyjęło się, że drugą potęgę obliczamy przy pomocy operatora mnożenia a nie potęgowania.

OCTAVE posiada znacznie więcej operatorów, niektóre pojawią się w dalszej części, inne są używane w dosyć specjalistycznym kontekście i nie bardzo jest sens o nich mówić. Natomiast warto wiedzieć, że niektóre znaki na klawiaturze mogą oznaczać operatory.

Jedną z cech OCTAVE sprawiających trudności początkującym jest fakt, że jako separatora dziesiętnego, czyli znaku oddzielającego część całkowitą od części ułamkowej, używa się znaku kropki a nie przecinka. Wynika to z faktu, że OCTAVE (jak też i inne języki programowania) powstała w kręgu kultury anglojęzycznej, gdzie używa się kropki jako separatora dziesiętnego.

Użycie w tym celu przecinka nie wywoła komunikatu o błędzie, ale znak przecinka jest pewnym operatorem. Dlatego efekt będzie raczej niezgodny z oczekiwaniami. Tak więc liczbę rzeczywistą zapisujemy:

---

<sup>2</sup>OCTAVE pozwala wpisać niekompletne wyrażenie a wtedy czeka, aż zostanie ono zakończone w kolejnych liniach.

3.14

**Test Knutha.** Jeśli spróbujemy wyliczyć wartość wyrażenia  $\left(\frac{4}{3} - 1\right) 3 - 1$  (znanego jako test Knutha):

$(4/3-1) * 3-1$

to w wyniku nie uzyskamy wartości zero, jak sugerowałaby nasza znajomość arytmetyki, ale wartość  $-2.2204e-16$ .

Najpierw zajmiemy się tym, co to znaczy, a następnie dlaczego tyle. Znaczenie takiej wartości wywodzi się od tzw. notacji naukowej dla liczb, czyli zapisu w postaci  $1.57 \cdot 10^{-5}$ . Notacja naukowa pozwala na dość elastyczne i wygodne zapisywanie zarówno bardzo dużych jak i bardzo małych liczb. W informatyce wygodą notacji naukowej jest bardzo pożądana, natomiast nie ma możliwości użycia jej wprost, z najprostszego powodu – ograniczeń klawiatury. W związku z tym przyjęto umowę, by liczbę w postaci  $1.57 \cdot 10^{-5}$  zapisywać jako  $1.57e-5$ , czyli najpierw wartość 1.57 a zamiast potęgi  $10^{-5}$  piszemy jedynie e-5. Obie części piszemy razem, bez znaku odstępu. Błędem jest także umieszczanie operatora mnożenia (\*), gdyż tutaj mnożenie jest częścią zapisu.

Teraz już wiemy, że OCTAVE jako wartość  $\left(\frac{4}{3} - 1\right) 3 - 1$  obliczy nie wartość zero ale  $-2.2204 \cdot 10^{-16}$ . Nie jest to błąd OCTAVE, każdy język i każdy program dadzą identyczny wynik<sup>3</sup>. Jest to efekt *skończonej reprezentacji* liczb rzeczywistych w trakcie przetwarzania. Komputer przeznaczony na każdą liczbę rzeczywistą taką ilość bitów jaką przewidzieli twórcy procesora<sup>4</sup>. Jeśli chcemy obliczyć wartość  $\frac{4}{3}$ , która wynosi 1.33(3) a więc ma nieskończenie wiele cyfr rozwinięcia dziesiętnego – binarnego także, to potrzeba na to nieskończenie wiele bitów do zapisania takiej liczby. Natomiast ilość bitów, które mamy do dyspozycji jest z góry określona. Tak więc dzieląc 4 przez 3 wcale nie otrzymujemy  $\frac{4}{3}$  ale przybliżenie tej wartości, które wynika ze skończonej liczby bitów przeznaczonych na reprezentację  $\frac{4}{3}$ . Dlatego w teście Knutha, po pomnożeniu *przybliżenia*  $\frac{4}{3}$  przez 3 wcale nie otrzymujemy wartości 4 a w konsekwencji wartość całego wyrażenia wcale nie równa się zero.

Warto zwrócić uwagę, że wynik zależy od kolejności operacji, obliczenie testu Knutha jako:  $\left(\frac{4}{3} \cdot 3 - 1 \cdot 3\right) - 1$  da prawidłowy wynik.

**Zero numeryczne.** Błędy zaokrąglenia wynikające ze skończonej reprezentacji liczb rzeczywistych są nieuniknioną konsekwencją takiej a nie innej budowy komputera.

Ich wpływ na wynik końcowy może być bardzo różny. Od żadnego po katastrofalny. Jak ocenić wpływ tych błędów na wynik mówi nam *Analiza numeryczna* a pokrewne jej *Metody numeryczne* szukają takich sposobów organizacji obliczeń (wynik zależy od kolejności) aby ten wpływ zminimalizować.

<sup>3</sup>O ile do obliczeń zostanie użyta wbudowana arytmetyka procesora. Sama wartość zależy od architektury procesora.

<sup>4</sup>Na platformie PC wszystkie procesory używają standardu arytmetyki IEEE 754, która przewiduje 32 lub 64 bity na liczbę rzeczywistą.

Wartość  $-2.2204 \cdot 10^{-16}$  jest wartością bardzo małą. Taką małą wartość, w stosunku do wielkości obliczanych, zowie się *zerem numerycznym*.

Czy dana wielkość jest czy nie jest zerem numerycznym, w dużym stopniu jest kategorią subiektywną.

{zero-maszynowe}

**Zero maszynowe.** W przeciwieństwie do zera numerycznego, traktowanego subiektywnie, pojęcie *zera maszynowego* jest obiektywne. Jest to taka największa liczba, która dodana do jedynki nadal daje jeden:  $1 + \varepsilon \rightarrow 1$ .

Podobną do zera maszynowego wielkością jest tzw. *precyzja maszyny*, czyli najmniejsza liczba, która dodana do jedynki daje wartość większą od jeden:  $1 + \varepsilon > 1$ .

Te dwie wielkości są sąsiednimi punktami na osi liczb reprezentowanych w arytmetyce zmiennoprzecinkowej<sup>5</sup>.

**Wartości specjalne** Arytmetyka IEEE 754, której używają także wszystkie procesory stosowane w PC, definiuje pewne wartości specjalne. Takie jak Inf (ze znakiem, czyli zarówno +Inf jak i -Inf), którą można uzyskać obliczając  $1/0$  oraz NaN.

Jakkolwiek Inf jest zbliżona w koncepcji do  $\infty$  to należy pamiętać, że arytmetyka procesora nie umie operować na takich symbolach. W związku z tym, te wartości służą jedynie do obsługi błędów.

Wartością specjalną jest również NaN – *Not A Number* – dosłownie: „to nie jest liczba”.

## 1.2 Stałe i funkcje

OCTAVE ma zdefiniowane różne przydatne stałe, przede wszystkim wartość (rozwiniecie dziesiętne) liczby  $\pi$ . Do tej wartości można odwołać się wpisując po prostu `pi`, tak jakbyśmy naciskali klawisz z napisem `pi`  
`pi/2`

OCTAVE jest wyposażona również w ogromną liczbę funkcji matematycznych, np. dobrze znaną funkcję  $\sin(x)$ . Obliczenie wartości funkcji zapisuje się w sposób zbliżony do zapisu matematycznego:

`sin(pi/2)`

Jedyna różnica polega na tym, że w matematyce często rezygnujemy z nawiasów przy argumentach, pisząc  $\sin \frac{\pi}{2}$ , gdyż jest to jednoznaczne, natomiast OCTAVE nie dopuszcza zapisu bez nawiasów.

Liczba funkcji matematycznych dostępnych w programie OCTAVE jest doprawdy imponująca. Oprócz wszystkich typowych funkcji matematycznych, powszechnie znanych, takich jak trygonometryczne czy logarytmy, dostępne są funkcje zna-

<sup>5</sup>Oś zwierająca liczby rzeczywiste jest ciągła ale przy skończonej ilości bitów, możemy reprezentować tylko niektóre z tych wartości, więc będą to punkty.

ne raczej w węższych kręgach specjalistów, dość egzotyczne dla przeciętnego człowieka. Takimi są np. rodziny funkcji hiperbolicznych  $\sinh$ ,  $\cosh$  włącznie z ich arcusami.

Nazwy niektórych funkcji mogą być zaskoczeniem. Z jednej strony ich zapis wynika z konwencji stosowanej w krajach anglojęzycznych, gdzie funkcja  $\tan$  jest zapisywana jako  $\tan$ , z drugiej strony część funkcji twórcy programu nazwali w sposób niezgodny z naszymi przyzwyczajeniami. Tak np.  $\log()$  to  $\log_e$  (czyli  $\ln$ ) a  $\log$  czyli  $\log_{10}$  to  $\log_{10}()$ , zaś  $\log_2$  to  $\log_2()$ .

Na koniec warto zwrócić uwagę na kilka dobrze znanych funkcji, które się inaczej zapisuje, głównie z uwagi na możliwości klawiatury. Pierwszą jest funkcja  $e^x$ , której nie da się w tej postaci wpisać z klawiatury, więc funkcja nazywa się  $\exp()$ . Nie powinno to być dużą nowością, czasami stosuje się w matematyce też zapis  $\exp(x)$ .

Drugą jest funkcja  $\sqrt{x}$ , której znowu z klawiatury nie da się wpisać, więc funkcja nazywa się  $\text{sqrt}()$ . Wpisując  $\text{sqrt}(4)$  powinniśmy otrzymać wynik 2 jako wartość  $\sqrt{4}$ .

Interesujące byłoby wpisanie żądania obliczenia  $\text{sqrt}(-4)$ . Wynikiem będzie nie komunikat o błędzie, ale wartość  $0 + 2i$ , czyli liczba zespolona. OCTAVE domyślnie operuje na liczbach zespolonych. Ponieważ liczby rzeczywiste są podzbiorem liczb zespolonych, więc prawie wszystkie wyniki będą zgodne z naszymi oczekiwaniami. Prawie, gdyż właśnie wartość  $\sqrt{-4}$  nie istnieje w dziedzinie liczb rzeczywistych, natomiast istnieje w dziedzinie liczb zespolonych.

Co prawda szanse są minimalne, ale może się tak zdarzyć, że błędnie wpisana operacja wykona się, gdyż OCTAVE używa innej (szerszej) definicji operatora bądź funkcji.

### 1.3 Zmienne

Typowo, używając kalkulatora, obliczamy pewną wielkość, która będzie użyta w dalszych obliczeniach. Kalkulatory często oferują możliwość zapamiętania jednej wielkości, która potem może być wykorzystana w dalszych obliczeniach. Niestety często się zdarza, że potrzebujemy policzyć kilka wielkości pośrednich na podstawie których wyliczamy wartość końcową. Wtedy pracowicie przepisujemy wyniki cząstkowe na kartkę a następnie wprowadzamy je z powrotem do kalkulatora.

Udogodnieniem oferowanym przez OCTAVE, w porównaniu z kalkulatorem, jest możliwość zapamiętywania dowolnej liczby wyników obliczeń pośrednich, bez konieczności przepisywania ich na kartkę i z powrotem.

Chcąc przechować taki wynik, trzeba mu najpierw nadać unikalną nazwę, aby w przypadku kilku takich wielkości, móc jednoznacznie określić, którą z nich mamy na myśli.

{zmienna}

Taką wartość do przechowania tworzymy i przypisujemy jej wartość za pomocą polecenia typu:

```
x=1
```

czyli w tym przypadku stworzyliśmy nową przechowywaną wartość pod nazwą  $x$  i umieściliśmy tam liczbę 1.

Polecenie to ma sporo złożonych aspektów, którym poświęcony jest cały następny rozdział, tutaj potraktujemy je na razie skrajnie prosto: tak to się robi.

W ten sposób zapamiętaliśmy pod nazwą  $x$  jakąś wartość, w tym wypadku stałą 1. Równie dobrze możemy zapamiętać wynik dowolnego wyrażenia arytmetycznego, np.:  $x = (2 + \sin(3 \cdot \pi)) / \log(30)$

Taką zapamiętaną wartość (zwaną zmienną, przez analogię do matematyki) możemy użyć w późniejszych obliczeniach  $\exp(\pi) \cdot 2 \cdot x$

Możemy również zdefiniować wartość innej zmiennej poprzez wartość poprzednio wyliczonej:  $z = 1/x$

Warto zwrócić uwagę, że przypisanie nie wyraża relacji. Mając funkcję,  $y(x) = ax^4 + bx^2 + c$ , chętnie używamy podstawienia:  $t = x^2$  i otrzymujemy równanie kwadratowe:  $y(t) = at^2 + bt + c$ , które łatwo rozwiązać.

Przypisanie:  $t = x \cdot x$  wpisze do zmiennej  $t$  kwadrat tego, co znajduje się w zmiennej  $x$ , ale po zmianie zawartości  $x$  zmienna  $t$  nie zostanie uaktualniona.

## 1.4 Skrypty

Wpisywanie i obliczenie wyrażeń po kolei jest dość wygodne dla krótkich sekwencji obliczeń. Kiedy mamy więcej poleceń (np. 20), to musimy liczyć się z tym, że się pomylimy i często w takim wypadku wypadałoby wpisywać wszystko od nowa.

W takiej sytuacji OCTAVE daje nam możliwość pracy w trybie nieinteraktywnym lub inaczej wsadowym.

Tryb ten polega na tym, że ciąg poleceń dla OCTAVE wpisujemy w pliku tekstowym (np. `skrypt.m`). Taki plik w terminologii UNIXa zwie się *skryptem*. Następnie uruchamiamy OCTAVE z parametrem w postaci nazwy pliku:

```
octave -q skrypt.m
```

W efekcie OCTAVE przeczyta każdą linię z pliku `skrypt.m` tak, jakby była przez nas wpisywana z klawiatury. Opcja `-q` (**q**uiet – cicho) nie jest niezbędna, zapobiega ona wyświetlaniu ekranu powitalnego, natomiast nie wpływa na wyniki.

Dzięki skryptom możemy wykonywać obliczenia wymagające tysięcy linii. Wpisanie nawet długiej sekwencji obliczeń do pliku tekstowego umożliwia, jeśli nawet się pomyliliśmy, ponowną edycję pliku, a po poprawieniu błędu<sup>6</sup> znowu zlecić OCTAVE wykonanie zawartości pliku.

Należy podkreślić, że oba tryby pracy: interaktywny i nieinteraktywny, nie wykluczają się wzajemnie a raczej uzupełniają. Dla krótkiego ciągu obliczanych wielkości szybciej i efektywniej jest wpisać to bezpośrednio, natomiast dla długich sekwencji zazwyczaj wygodniej jest umieścić wszystkie instrukcje w pliku.

<sup>6</sup>Stare powiedzenie testerów: po znalezieniu i poprawieniu dowolnej liczby błędów i tak zostanie jeszcze jeden.



**Komentarze.** Szczególnie w przypadku skryptów warto używać komentarzy. Znak procentu % jest znakiem komentarza. Od momentu jego wystąpienia do końca linii OCTAVE ignoruje wszystkie znaki tak, jakby ich tam nie było.

`a=3 % stała w równaniu`

Z punktu widzenia OCTAVE komentarze są niepotrzebne. One są potrzebne piszącemu, aby zajrzawszy do pliku np. po dwu tygodniach szybko zorientował się w zasadniczej myśli programu.

## 1.5 Wypisywanie

Szczególnie w przypadku pracy nieinteraktywnej i przy dłuższych obliczeniach, często niewygodne staje się wyświetlanie każdego wyniku przez OCTAVE.

Możemy sobie zażyczyć, aby wynik dowolnego obliczenia nie był wyświetlany umieszczając znak średnika ; na końcu linii. Przykładowo instrukcja:

```
x=1+1;
```

umieści w zmiennej x wartość 2, ale nie wypisze tej wartości na ekranie. Aby przekonać się, że w x rzeczywiście jest 2, wystarczy wpisać w linii nazwę zmiennej:

```
x
```

Ten sposób czasami bywa niedogodny, więc OCTAVE oferuje też inną możliwość wypisywania wartości, za pomocą funkcji `disp` (ang. *display* – wyświetl). Tak więc polecenie:

```
disp(x)
```

wypisze wartość zmiennej x. Oba sposoby różnią się nieco postacią w jakiej wartość będzie wypisywana.

Funkcja `disp` umożliwia też wypisywanie tekstów w trakcie wykonywania programu:

```
disp("Witamy w świecie programowania");
```

Oprócz funkcji `disp()` OCTAVE umożliwia korzystanie z funkcji `printf`, analogicznej do funkcji języka C. Funkcja ta jest o wiele bardziej złożona w użyciu, jednak umożliwia całkowitą kontrolę nad sposobem wypisywania.

Na koniec, do kompletu, warto wspomnieć, że gdy domyślny sposób wyświetlania liczb jest zbyt mało dokładny, magiczne polecenie `format long` pozwala go zmienić na bardziej precyzyjny.

## 1.6 Własne funkcje

Jak już zaznaczono, OCTAVE posiada praktycznie wszystkie funkcje matematyczne, które mogłyby być potrzebne, jednak zawsze okaże się, że potrzebujemy jeszcze jednej. OCTAVE umożliwia pisanie swoich własnych funkcji.

Aby zdefiniować własną funkcję, należy na początku zaznaczyć, że ten fragment jest początkiem definicji. Służy do tego słowo kluczowe `function`. Następnie (w tej samej linii) wpisujemy pod jaką nazwą zmiennej (np. `y`) będzie umieszczony obliczony wynik, potem (po znaku `=`) nazwa funkcji (np. `foo`). W nawiasach

okrągłych umieszczamy nazwę argumentu (np.  $x$ ), czyli wielkości dla której mamy obliczyć wartość. W sumie, pierwsza linia wyglądałaby:

```
function y=foo(x)
```

Po deklaracji następuje część zwana *ciałem funkcji*, gdzie zawarty jest ciąg poleceń, który dla zadanej wartości argumentu obliczy wynik.

Ciało funkcji może zawierać dowolną liczbę linii (instrukcji). Od zerowej, co nie ma sensu ale jest poprawne z punktu widzenia OCTAVE, do milionów linii.

Zazwyczaj, z czysto praktycznych względów, staramy się, aby funkcja mieściła się w oknie edytora, gdyż łatwo wtedy ogarnąć wzrokiem całość.

Funkcję kończy słowo kluczowe `end%function`<sup>7</sup>.

Przykładem najprostszej funkcji może być:

```
1 function y=foo(x)
2     y=x+1;
3 end%function
```

Nasza przykładowa funkcja nazywa się `foo` a dokonuje prostego obliczenia, do wartości argumentu dodaje jeden.

Od tej pory możemy używać naszej funkcji `foo` tak, jak każdej innej, czyli np. :  
`x=foo(3)`

Funkcja musi być zdefiniowana przed pierwszym użyciem, czyli najpierw musi wystąpić część `function` aż do `end%function` a dopiero potem można jej użyć.

Zazwyczaj własne funkcje definiujemy przy trochę bardziej złożonych obliczeniach, dlatego wygodniej jest umieszczać te obliczenia w pliku. Wtedy funkcja (bądź funkcje – może ich być więcej) są na początku pliku a potem następują obliczenia, odwołujące się do już zdefiniowanych funkcji. Taki skrypt, zawierający funkcję(e) oraz jej (ich) wywołanie będziemy nazywali umownie „programem”, chociaż nie jest to najwłaściwsza nazwa. Jednak pozwoli nam to uniknąć wielu nieporozumień.

Z uwagi na mechanizm plików funkcyjnych, opisanych w pkt. 1.7, plik z programem nie powinien nazywać się tak, jak pierwsza funkcja w pliku.

## 1.7 \*Uruchamianie

{plikf}

Istnieje bardzo wiele sposobów uruchamiania skryptów OCTAVE, w szczególności warto wspomnieć o tzw. *plikach funkcyjnych*. Plik funkcyjny zawiera jedynie jedną funkcję i nazywa się tak jak funkcja z końcówką `.m`. Zatem plik zawierający

<sup>7</sup>Tu dochodzimy do jedynej istotnej różnicy między `matlabem` i `OCTAVE`. `Matlab` wymaga słowa kluczowego `end`, `OCTAVE` oprócz `end` pozwala umieszczać `endfunction` co ma ogromne zalety, gdyż widać, co dany `end` zamyka. Wersja zapisu `end%function` godzi oba podejścia. Jest akceptowalna przez `matlab` a jednocześnie pokazuje co zamykamy.

funkcję `foo` musi się nazywać `foo.m`. Jeśli plik ten znajduje się w bieżącej kartoce to OCTAVE wie, gdzie jest definicja funkcji `foo`. Dokładniej, to odbywa się to odwrotnie, jeśli odwołamy się do jakiejś funkcji (np. `ala`) a OCTAVE nie ma jej w pamięci, to przeszukuje zarówno biblioteki dynamiczne, jak też przeszukuje kartotekę bieżącą (tą z której uruchomiono OCTAVE), w poszukiwaniu pliku `ala.m`. Jeśli znajdzie taki plik (i zawiera on funkcję `ala`), to jest ona wykonywana.

Jest to sposób bardzo wygodny przy realizacji większych projektów i dlatego często stosowany. Ponieważ celem tego kursu jest nauka elementarnych podstaw, Autor uważa, że do tego celu wygodniej jest używać osobnych plików zawierających zarówno funkcję, jak i jej wywołanie.

Należy unikać nadawania plikom rozszerzenia `.oct`. To rozszerzenie jest zarezerwowane dla plików binarnych zawierających dynamicznie ładowalne funkcje OCTAVE. Używanie takiego rozszerzenia może wywoływać tzw. skutki uboczne i powodować kompletnie niezrozumiałe zachowanie OCTAVE.

## 1.8 Ćwiczenia

1. Obliczyć wartość wyrażenia:  $\frac{5^2+1}{5^3+2\cdot 5^2-3\cdot 5+7}$
2. Obliczyć wartość wyrażenia  $\sqrt{1 - \sin^2\left(\frac{\pi}{15}\right)}$
3. Napisać wyrażenie obliczające wartość ułamka łańcuchowego:

$$\frac{1}{1 + \frac{1}{1 + \frac{1}{2}}}$$

4. Obliczyć wartości:
  - (a) `1/0`
  - (b) `-1/0`
  - (c) `0/0`
  - (d) `Inf+5`
  - (e) `Inf/21`
  - (f) `Inf-Inf`
  - (g) `NaN+5`
  - (h) `NaN/21`
  - (i) `NaN+NaN`
5. Pokazać różnicę przy wświetlaniu wartości  $\frac{1}{3}$  powodowaną przez dyrektywę `format long`
6. Wypisać komunikat: „Serdecznie witamy!!”.

Wersja 2010

Wersja: 4 października 2010

## Rozdział 2

# Instrukcja przypisania

{chap:przypis}

**Nazwy zmiennych.** Nazwy zmiennych są dosyć dowolne, mogą składać się z liter (wielkie i małe są rozróżniane!), cyfr i znaku podkreślenia, przy czym nie mogą zaczynać się od cyfry. Mogą być jednoznakowe. Istnieje bardzo wiele konwencji nazywania zmiennych, każdy programista ma swoje obyczaje. Ponieważ dla OCTAVE jest obojętne jak zmienne się nazywają, więc istotne jest to, aby nazwy sugerowały znaczenie tego, co w danej zmiennej jest przechowywane. Przykładowo, jeśli obliczamy wyróżnik równania kwadratowego  $\Delta$ , to można zmienną nazwać D lub Delta albo delta. Taka nazwa wyraźnie sugeruje, co jest w tej zmiennej, w przeciwieństwie do nazwy p – która równie dobrze mogłaby zostać tutaj użyta.

Jak już wcześniej (str. 13) wspomniano, zmienna pozwala przechować w pamięci obliczoną wartość do późniejszego użycia. Zmienna w OCTAVE istnieje od momentu, kiedy po raz pierwszy nadamy jej wartość. W tym momencie rezerwowany jest w pamięci komputera obszar niezbędny do przechowania tej wartości a nazwa zmiennej służy jako etykieta (adres), która umożliwia odczytanie czy zapisanie w odpowiednim miejscu pamięci. Przy zmianie wartości zmiennej czyli np.  $x=2$ , cokolwiek było w miejscu wskazywanym przez etykietę  $x$  jest nadpisywane i nie mamy możliwości odzyskania poprzedniej wartości.

Jest to pełna analogia do sposobu działania pamięci w kalkulatorze.

**Znak =** Wbrew pozorom linia typu

$x=1$

nie ma nic wspólnego z równaniem

$$x = 1$$

W kategoriach matematyki, jeśli prawa strona równa się lewej to lewa strona równa się prawej, czyli  $1 = x$ . Jeśli spróbujemy wpisać do OCTAVE linie

$1=x$

to otrzymamy komunikat o błędzie.

Równanie matematyczne  $y = x + 1$  przedstawia na płaszczyźnie zbiór nieskończenie wielu punktów – prostą, tymczasem instrukcja  $y=x+1$  nie narysuje żadnej prostej a żądanie wyświetlenia wartości  $y$  pokaże nam jedną wartość.

Znak = w tych wszystkich instrukcjach jest mylący, gdyż nie jest to symbol relacji równości matematycznej ale **operator przypisania**.

W wielu książkach opisujących programowanie operator przypisania zapisuje się za pomocą strzałki  $\leftarrow$ , co oznacza „wpisz”. Pomimo, że taki zapis byłby bardzo przejrzysty i nie prowadziłby do nieporozumień, to nie da się wpisać takiego znaku do pliku, gdyż nie ma go na klawiaturze.

W pierwszych wersjach Basic-u używano zapisu `LET x 1`, czyli niech  $x$  będzie 1. W języku Pascal – przeznaczonym do nauki programowania – używa się operatora `: =`. Obecnie, we wszystkich językach programowania (w których używa się zapisu operatora przypisania) powszechnie używa się znaku `=`, zakładając, że programiści wiedzą, że nie jest to matematyczna relacja równości tylko operator przypisania.

{przypisanie}

**Operator przypisania.** Instrukcja w skład którego wchodzi wyrażenie przypisania ma bardzo precyzyjnie określoną budowę. Mówi się o wartości lewostronnej i wartości prawostronnej.

Po lewej stronie operatora przypisania (znaku `=`) może znajdować się jedynie nazwa zmiennej.

Po prawej stronie operatora przypisania może znajdować się dowolnie złożone wyrażenie, byle dało się wyliczyć jego wartość.

Instrukcja jest wykonywana w ten sposób, że wyliczana jest wartość prawostronna (czyli wylicza się, ile wynosi to, co jest po prawej stronie) a następnie do zmiennej, której nazwa znajduje się po lewej stronie, wpisywana jest wartość prawostronna.

Najprostszym przykładem wyrażenia jest stała:

`x=1`

Czyli wartością prawostronną jest 1 a wynikiem tej instrukcji jest wpisanie wartości 1 do zmiennej (komórki pamięci)  $x$ .

Bardziej złożonym przykładem jest:

`y=sin(pi/4)*cos(7*pi/3)`

gdzie wyliczana jest wartość wyrażenia  $\sin(\frac{\pi}{4})\cos(\frac{7\pi}{3})$ , która jest wartością prawostronną (a wynosi 0,35355) i ta jest wpisywana do komórki  $x$ .

Używając operatora przypisania można odwoływać się do wartości już wcześniej zdefiniowanych (istniejących zmiennych). Czyli jeśli napiszemy:

`y=sin(pi/4)*cos(7*pi/3)`

`z=1/(y*y+1)`

to wartość wpisana do komórki o nazwie  $z$  będzie wynosiła 2,8284, gdyż najpierw do zmiennej (komórki pamięci)  $y$  zostanie wpisane 0,35355 a następnie, w kolejnej instrukcji, przy obliczaniu wartości prawej strony zostanie pobrana zawartość zmiennej  $y$  i użyta do obliczeń.

Należy jednak podkreślić, że ponieważ prawa strona musi dać się wyliczyć, więc nie można po prawej stronie użyć zmiennej, która nie istnieje (nie przypisano jej żadnej wartości). Tak więc wpisanie instrukcji:

$z=1/(y*y+1)$

jako pierwszej instrukcji w programie (lub w sesji) zakończy się niemiłym komunikatem o błędzie. Sens tego komunikatu jest taki, że odwołano się do niezdefiniowanej wartości  $y$ .

Najbardziej zaskakującą konsekwencją takiego działania operatora przypisania jest możliwość użycia go po obu stronach wyrażenia. O ile wcześniej została nadana zmiennej (tutaj używamy  $x$ ) wartość, to możemy napisać:

$x=x+1$

W kategoriach równania matematycznego  $x = x + 1$  jest równaniem sprzecznym, gdyż nie ma takiej wartości  $x$ , któraby to równanie spełniała. Gdyby takie  $x$  istniało, to musiałyby zachodzić  $0 = 1$ <sup>1</sup>. Natomiast taka instrukcja wykona się, gdyż oznacza ona, że należy najpierw wziąć tą wartość, która jest w zmiennej (komórce pamięci)  $x$ , dodać do tego jedynkę i tak otrzymany wynik wpisać do tejże samej zmiennej  $x$ .

{przypisanie-rekur}

**Operator przypisania a matematyka.** Gdybyśmy chcieli wyrazić przypisanie, (np.  $x=x+1$ ) w kategoriach równania matematycznego to należałoby napisać:

$$x_{n+1} = x_n + 1$$

czyli wartości  $x$  po obu stronach równania to dwie różne wielkości. Miałoby to taki sens, że w komórce pamięci o etykiecie  $x$  zapisywane są różne wielkości, najpierw jakaś  $x_1$ , potem inna  $x_2$  i kolejne. Tak więc wartość następna ( $n$  plus pierwsza) jest równa obecnej ( $n$ -tej) powiększonej o wartość 1.

Ta analogia jest o tyle pożyteczna, że działa również w drugą stronę. Jeżeli mamy oprogramować wzór matematyczny typu:

$$x_{n+1} = f(x_n)$$

to w kategoriach operatora przypisania należałoby napisać:

$x=f(x)$

**Koło Mohra** Dla ilustracji jak opisane zasady pozwalają nam dokonywać praktycznych obliczeń, rozważmy zadanie tzw. koła Mohra. Koło Mohra pozwala na graficzne znalezienie ważnych wielkości dla stanu naprężenia<sup>2</sup>. Dla zadanych trzech wartości:  $\sigma_x$ ,  $\sigma_y$  i  $\tau_{xy}$  pozwala wyznaczyć naprężenia główne  $\sigma_1$ ,  $\sigma_2$  oraz maksymalne naprężenie styczne  $\tau_{\max}$  (które jest osiągane w innym układzie niż naprężenia główne).

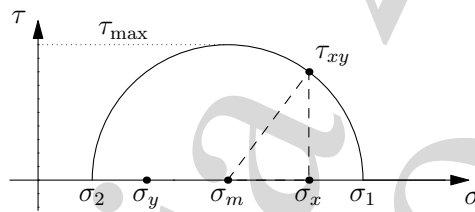
Konstrukcja przebiega następująco:  $\sigma_x$  i  $\sigma_y$  odkładamy na osi  $\sigma$ .

Następnie wyznaczamy wartość średnią  $\sigma_m$  jako średnią arytmetyczną  $\sigma_x$  i  $\sigma_y$ , punkt  $\sigma_m$  jest w połowie odcinka  $\sigma_x \sigma_y$ .

Z punktu  $\sigma_x$  odkładamy wartość  $\tau_{xy}$  prostopadłe do osi. Zataczamy okrąg z punktu  $\sigma_m$  przez  $\tau_{xy}$ . Punkty przecięcia tego okręgu z osią  $\sigma$  dają nam  $\sigma_1$  (większa

<sup>1</sup> Wystarczy odjąć od obu stron  $x$ .

<sup>2</sup> Czymkolwiek ten stan naprężenia jest.



Rysunek 2.1: Koło Mohra.

{fig:mohr}

wartość) i  $\sigma_2$  (mniejsza). Promień okręgu określa wartość  $\tau_{\max}$ .

Opisaną konstrukcję geometryczną można wyrazić wzorami:

$$\sigma_m = \frac{\sigma_x + \sigma_y}{2}$$

$$\tau_{\max} = \sqrt{(\sigma_x - \sigma_m)^2 + \tau_{xy}^2}$$

$$\sigma_{1,2} = \sigma_m \pm \tau_{\max}$$

{mohr} Napişmy program:

```

1 % dane umieszczamy na początku, łatwo wtedy
2 % przekształcić program w funkcję
3 sx= 3.15;
4 sy= 1.72;
5 t_xy= 2.3;

6 sm= (sx+sy)/2;

7 % długość podstawy trójkąta
8 p=sx-sm;
9 % promień okręgu
10 r=sqrt(p*p + t_xy*t_xy);

11 s1= sm+r;
12 s2= sm-r;
13 t_max= r;

14 disp("Max. naprężenie główne");
15 disp(s1);

16 disp("Min. naprężenie główne");
17 disp(s2);

18 disp("Max. naprężenie styczne");
19 disp(t_max);

```



`{sec:swap}`

**Wymiana zawartości dwu zmiennych.** Czasami zachodzi potrzeba aby zamienić wartości wpisane do zmiennych czyli aby to, co było w jednej znalazło się drugiej i odwrotnie. Takie zadanie wymaga użycia trzeciej zmiennej (pomocniczej).

Jeśli mamy dwie zmienne  $x$  i  $y$ , to następująca sekwencja poleceń realizuje wymianę wartości używając pomocniczej zmiennej  $tmp$ :

`{swap}`

```
1  x=1.2;
2  y=2.53;

3  tmp=y;
4  y=x;
5  x=tmp;

6  disp(x);
7  disp(y);
```

### \*Typy.

`{sec:strcat}`

**Zaawansowane wypisywanie.** Warto zilustrować cechy wyrażenia przypisania na przykładzie wypisywania wartości. Te przykłady będą nam potrzebne w dalszych programach.

Generalnie OCTAVE pozwala nam ignorować typy zmiennych, gdyż automatycznie przeprowadza konwersję typów w razie potrzeby. Programując, nie potrzebujemy pamiętać o typach zmiennych a przynajmniej tak by się mogło wydawać. Jednym z wyjątków od tej zasady jest wypisywanie wielkości.

Ciąg znaków<sup>3</sup> "123" różni się od liczby 123 tym, że ta pierwsza wielkość składa się z osobnych znaków 1, 2 i 3. W takim ciągu równie dobrze mogłyby się pojawić inne znaki dostępne na klawiaturze: "a123!2\_5!x67". W przypadku liczby 123, ta może składać się wyłącznie z cyfr od 0 do 9<sup>4</sup>

Do liczb możemy stosować wszystkie operatory arytmetyczne, oczywisty jest sens wyrażenia  $123 * 25$ . Łudząco podobne wyrażenie "123" \* "25" dla OCTAVE jest jakościowo tym samym co "a123!2\_5!x67" \* "as". Co mogłoby znaczyć mnożenie dwóch tekstów? Operacje dla ciągu znaków są inne niż operacje dla liczb<sup>5</sup>. W konsekwencji musimy rozróżniać, kiedy mamy do czynienia z liczbą a kiedy z ciągiem znaków (tekstem).

Funkcja `disp(123)` zamienia liczbę 123 na ciąg znaków (napis) "123" i dopiero wtedy wypisuje. Jednak podstawową wadą funkcji `disp()` jest fakt, że dopisuje

<sup>3</sup>Zwany również tekstem lub napisem, ang. *string*.

<sup>4</sup>Jak również mogą się pojawić: znak czyli  $\pm$ , separator dziesiętny (.) i literka e jako oznaczenie wykładnika.

<sup>5</sup>W OCTAVE jest sporo operacji, które można stosować do ciągu znaków ale nie będziemy ich opisywać.

znak nowej linii na końcu.

Gdybyśmy mieli zmienną *x* i chcieliby wypisać komunikat typu zawierający zarówno opis, jak i aktualną wartość zmiennej, np. Wartość *x*=123 to polecenia:

```
1 x=123;
2 disp("Wartość x=");
3 disp(x);
```

wypiszą:

```
Wartość x=
123
```

czyli opis będzie w jednej linii a wartość w następnej. Nie jest to na ogół to, co chcieliśmy uzyskać.

Jeśli chcemy wypisać komunikat zawierający zarówno ciągi znaków (tekst), jak i liczby w jednej linii, to musimy najpierw liczby zamienić na tekst a następnie połączyć wszystkie fragmenty tekstu w jedną linię i dopiero wtedy wypisać całość. Do tego potrzebne są dodatkowe funkcje.

Funkcja *num2str* – *number to string* – zamienia liczbę na tekst. Polecenie:

```
t=num2str(123)
```

zamieni liczbę 123 na tekst "123" i umieści go w zmiennej *t*. Warto zwrócić uwagę, że zmienna *t* zawiera teraz ciąg znaków i błędem jest stosowanie w stosunku do *t* operatorów właściwych dla liczb.

Funkcja *strcat* – *string concatenate* – łączy (dodaje) dwa lub więcej ciągi znaków. Linia:

```
w=strcat("x=", "3.14");
```

zapisze w zmiennej *w* ciąg znaków "x=3.14".

Program wypisujący komunikat mógłby mieć postać:

```
1 x=123;
2 b=num2str(x);
3 linia=strcat("Wartość x=",b);
4 disp(linia);
```

Warto dodać, że „sklejanie” za pomocą *strcat* ciągu "ala" i "as" daje w wyniku "alaas" a nie jak byśmy chcieli "ala as" a więc, jeśli chcemy uzyskać odstęp, to trzeba jawnie dodać znak odstępu " " w środku: *strcat*("ala", " ", "as").

Zamiast stałych możemy zawsze używać zmiennych. Jeśli, w pewnym momencie, zmienna *linia* zawiera napis "Ala" a zmienna *w* zawiera "ma asa" to polecenie *linia=strcat(linia," ", w)*; obliczy wartość prawostronną czyli „doklei” na końcu znak odstępu i napis "ma asa" dając "Ala ma asa" a następnie wpisze to do zmiennej *linia* jako nową jej wartość.

Używając tych funkcji możemy nie tylko dopisywać coś na końcu tekstu ale również na początku. Jeśli zamiast

```
linia=strcat(linia," ", w)
```

wpiszemy

```
linia=strcat(w," ", linia)
```

to jako wynik uzyskamy "ma asa Ala".

## 2.1 Ćwiczenia

1. Obliczyć  $x = r \cos(\phi)$  i  $y = r \sin(\phi)$  dla wartości  $r = 7,5$  i  $\phi = \frac{\pi}{7}$
2. Obliczyć wartość  $y = \sqrt{1 - \sin^2(\frac{\pi}{15})}$  używając podstawienia  $t = \sin(\frac{\pi}{15})$
3. Napisać program, który wypisuje wartości dwu zmiennych wraz z opisem w jednej linii.
4. Napisać funkcję, która wypisuje wartość zmiennej wraz z opisem.

Wersja 2010

Wersja: 4 października 2010